UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

PLGRAM

Presentado por Víctor Renuncio Tobar
en Universidad de Burgos — 6 de julio de 2016
Tutor: Dr. César Ignacio García Osorio
D. Álvar Arnaiz González

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

El Dr. César Ignacio García Osorio, profesor del departamento de Ingeniería Civil, área de Lenguajes y Sistemas Informáticos y D. Álvar Arnaiz González, profesor del departamento de Ingeniería Civil, área de Lenguajes y Sistemas Informáticos

Exponen:

Que el alumno D. Víctor Renuncio Tobar, con DNI 71.264.303-E, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado PLGRAM.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección de los que suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 6 de julio de 2016

Vº. Bº. del Tutor:

D. César Ignacio García Osorio

Vº. Bº. del co-tutor:

D. Álvar Arnaiz González

Resumen

Desarrollo de una aplicación que a partir de una gramática obtiene el análisis sintáctico descendente $LL(1)$ y los análisis sintácticos ascendente $SLR(1)$, $LR(1)$ y $LALR(1)$. Se obtienen los conjuntos *FIRST* y *FOLLOW* así como la tabla de análisis sintáctico predictivo (*TASP*) y las tablas de *ACCIÓN y de IR A*. Asimismo se puede obtener para los distintos análisis la traza de una cadena.

Con la aplicación se puede generar un archivo .XML para importar a la plataforma *Moodle* con la finalidad de generar de forma automática cuestionarios de análisis sintáctico. También se puede generar un fichero en formato LaTeX que se puede utilizar para realizar pruebas escritas. Se puede obtener un documento en el que el cuestionario esté completo, con las respuestas correctas y otro documento en el que el cuestionario está vacío, pensado para ser respondido por un tercero.

Descriptores

Generación de cuestionarios, análisis sintáctico ascendente, análisis sintáctico descendente, $LL(1)$, $SLR(1)$, $LALR(1)$, $LR(1)$.

Abstract

Development of an application that obtains top-down parsing $LL(1)$ and botton-up parsing $SLR(1)$, $LR(1)$ and $LALR(1)$ from a grammar. The application obtains the $FIRST$ and $FOLLOW$ sets and the predictive parsing table and tables $ACTION$ and $GOTO$. Moreover, it can be obtained for each diferent analyzes the trace of a string.

With the application you can generate a .XML file to import to the Moodle platform in order to generate automatically parse test. You can also generate a LaTeX file format that can be used for written tests. You can get a document that the questionnaire is completed, with the correct answers and other documents in which the questionnaire is empty, designed to be solved by students.

Keywords

Questionnaire generation, bottom-up parsing, top-down parsing, $LL(1)$, $SLR(1)$, $LALR(1)$, $LR(1)$.

Índice general

Índice general III

Índice de figuras V

Introducción 1
1.1. Introducción 1

Objetivos del proyecto 3
2.1. Objetivos técnicos 3
2.2. Objetivos académicos 4
2.3. Objetivos personales 4

Conceptos teóricos 5
3.1. Introducción 5
3.2. Análisis sintáctico 7
3.3. Análisis sintáctico descendente 8
3.4. Análisis sintáctico ascendente 11

Técnicas y herramientas 17
4.1. Técnicas 17
4.2. Herramientas 17

Aspectos relevantes del desarrollo del proyecto 23
5.1. Conocimientos necesarios 23
5.2. Conocimientos adquiridos 24
5.3. Objetivos realizados 24

Trabajos relacionados 25
6.1. PLQUIZ [2] 25
6.2. BURGRAM [9] 26

6.3. SEFALAS [4] 27
6.4. ANAGRA [7] 28
6.5. Proyecto SEPa [5] 29

Conclusiones y Líneas de trabajo futuras **31**
7.1. Conclusiones 31
7.2. Líneas de trabajo futuras 31

Bibliografía **33**

Índice de figuras

3.1. Estructura de un compilador [8]. 6
3.2. Estructura del analizador descendente [8]. 8
3.3. Tabla de análisis sintáctico predictivo [8]. 9
3.4. *TASP* de la gramática. 9
3.5. Traza de la cadena «`tipo begin codigo end $`». 10
3.6. Estructura del analizador ascendente. 12
3.7. Conjuntos de ítems $LR(0)$. 13
3.8. Primeros conjuntos de *ítems* $LR(1)$. 14
3.9. Tabla de ACCIÓN y de IR A en un análisis $LR(1)$. 14
3.10. Traza de análisis de la cadena «`tipo begin codigo end` $». . . . 15
6.11. PLQUIZ 26
6.12. BURGRAM 27
6.13. SEFALAS 28
6.14. ANAGRA 28
6.15. SEPa 29

Introducción

1.1. Introducción

Una de las principales características del actual concepto de docencia, sobre todo universitaria, es el gran auge que ha experimentado la docencia online. Esto se ve reflejado no sólo en el apoyo que supone a la docencia presencial, sino en la demanda de educación a distancia que cada vez tiene un mayor público.

Un claro ejemplo es el campus virtual de la Universidad de Burgos. UBUVirtual es una plataforma virtual basada en WebMoodle, que a su vez es una plataforma de eLearning utilizada por una gran cantidad de usuarios.

Por tanto, dotar de todas las herramientas necesarias a dicha plataforma para ofrecer un soporte útil y dinámico tanto al alumno como al profesorado, es una de las prioridades a tener en cuenta. Surge así la necesidad de realizar pruebas y exámenes a través de la plataforma.

En este proyecto se presenta una aplicación de escritorio que permite generar cuestionarios sobre análisis ascendente y descendente. Además, dicha aplicación permite su posterior publicación en Moodle y en formato LaTeX para su futuro uso en pruebas escritas.

Objetivos del proyecto

2.1. Objetivos técnicos

Los principales objetivos técnicos perseguidos mediante el desarrollo de la aplicación han sido:

1. Construcción de una GUI (Interfaz Gráfica de Usuario) que permita visualizar de forma rápida la aplicación desarrollada.
2. Construcción de una CLI (Command Line Interface) que permita trabajar de forma rápida y directa con la aplicación desarrollada.
3. Procesado y manipulación de gramáticas mediante procesadores del lenguaje.
4. Obtención del $FIRST$ y del $FOLLOW$ de los no terminales de una gramática.
5. Obtención de la Tabla de Análisis Sintáctico Predictivo ($TASP$) de una gramática.
6. Obtención de los conjuntos de items $LR(0)$ y $LR(1)$.
7. Obtención de las tablas de análisis sintáctico *ACCIÓN e IR A* para análisis $SLR(1)$, $LR(1)$ y $LALR(1)$.
8. Exportación de cuestionarios en formato *XML* compatible con la plataforma Moodle.
9. Exportación de cuestionarios en formato LaTeX.
10. Creación de un manual de usuario detallando todas las opciones de la aplicación.

2.2. Objetivos académicos

Tal y como se indica en la guía docente del Trabajo Fin de Grado, algunos de los objetivos académicos que se pretenden alcanzar con el mismo son los siguientes:

1. Desarrollo de un trabajo personal donde se apliquen los conocimientos teóricos y prácticos adquiridos en la titulación.
2. Desarrollo de la capacidad creativa mediante el planteamiento y resolución de un problema real.
3. Uso y aprendizaje de nuevas herramientas y aplicaciones vinculados a la Ingeniería Informática.
4. Desarrollo de la capacidad de exposición en público y defensa y argumentación del trabajo realizado.
5. Comunicar correctamente en otro idioma un resumen coherente del trabajo realizado.

2.3. Objetivos personales

El uso de VersionOne[1] me ha permitido organizar y mantener un control sobre las tareas que debía realizar. He realizado sesiones (*sprint*) de una semana que han coincidido con las reuniones con los tutores.

El uso de GitHub[2] me ha permitido llevar el control de versiones desde el inicio del trabajo.

He podido profundizar en el funcionamiento de ciertos aspectos de la máquina virtual de Java: sus librerías nativas, sus métodos de carga dinámica de librerías, paquetes y clases y sus efectos específicos sobre procesos ya en ejecución.

He aprendido a aprovechar las ventajas del lenguaje de programación Java[3]: mantener su capacidad multiplataforma evitando realizar llamadas al sistema operativo, sacar partido de sus bibliotecas nativas para la carga de objetos externos y lectura de objetos internos compilados y también explorar las posibilidades de librerías Java de licencia gratuita creadas por su inmensa comunidad de desarrolladores.

[1] http://www.VersionOne.com/
[2] http://www.GitHub.com/
[3] http://www.java.com.es/

Conceptos teóricos

3.1. Introducción

Todo el software que se ejecuta en las computadoras se escribe en un algún lenguaje de programación. Pero antes de poder ejecutar el programa, primero debe transformarse a un formato en el que la computadora pueda ejecutarlo.

Un *compilador* es un programa que lee un programa (*programa fuente*) escrito en un lenguaje (*lenguaje fuente*), y lo traduce a un programa equivalente (*programa objeto*) en otro lenguaje (*lenguaje objeto*).

El proceso de compilación se puede dividir en varias etapas: *ANÁLISIS* (*front end* o *etapa inicial*), *generación de código intermedio* (*middle end*) y *SÍNTESIS* (*back end* o *etapa final*). La etapa de *ANÁLISIS* divide el programa fuente en componentes e impone una estructura gramatical sobre ellas. También recoge información sobre el programa fuente que se almacena en una estructura de datos llamada *tabla de símbolos*. Después se utiliza esta estructura gramatical para crear una representación intermedia del programa fuente. La etapa de *SÍNTESIS* construye el programa destino a partir de la representación intermedia y de la información de la *tabla de símbolos*[3][12].

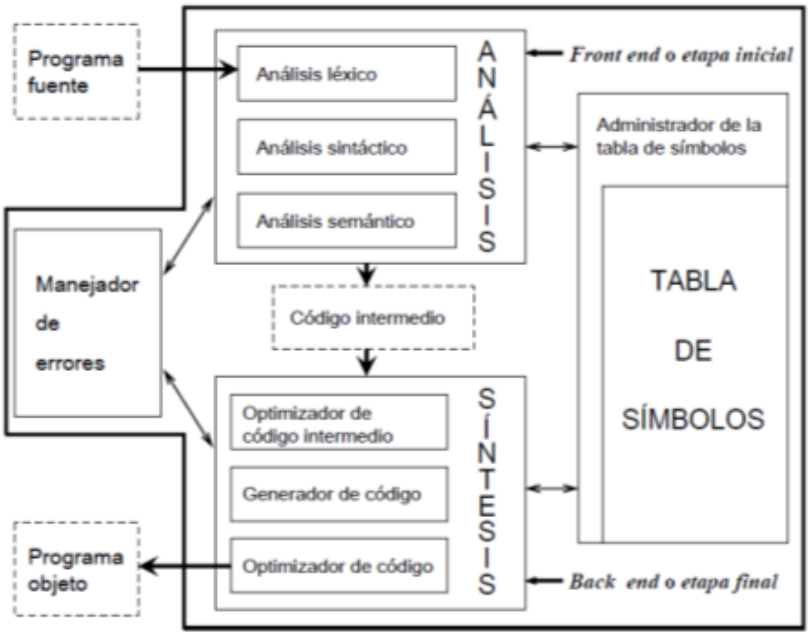

Figura 3.1: Estructura de un compilador [8].

La etapa de *ANÁLISIS* agrupa aquellas fases que dependen principalmente del lenguaje fuente y comprende:

- El **analizador léxico** (*scanner o lexer*) que agrupa los caracteres individuales en entidades lógicas.
- El **analizador sintáctico** (*parser*), que trataremos con más detalle a continuación, analiza la estructura general de todo el programa, agrupando las entidades simples identificadas por el *scanner* en construcciones mayores como sentencias, bucles, rutinas, que componen el programa complejo. Normalmente se utiliza la representación de *árboles sintácticos* para reflejar dicha estructura.
- El **analizador semántico** utiliza los *árboles sintácticos* y la información de la *tabla de símbolos* para comprobar la consistencia semántica del programa. También recopila información sobre el tipo (qué variables almacenan enteros, cuáles número en coma flotante,...).

La etapa de *SÍNTESIS* agrupa aquellas fases que dependen principalmente de la máquina objetivo y comprende:

- El **optimizador de código intermedio**, que transforma la representación intermedia en otra equivalente pero más eficiente.
- El **generador de código**, que genera un programa equivalente para su ejecución en la máquina objetivo, añadiéndole posiblemente rutinas de biblioteca y código de inicialización.
- Puede haber un **optimizador** de código para mejorar aún más el código generado.

3.2. Análisis sintáctico

El análisis sintáctico (*parser*) es la fase del procesador de lenguaje que toma como entrada los componentes léxicos (*tokens*) que le envía el analizador léxico y comprueba si con ellos se puede formar alguna sentencia válida del lenguaje[1].

La sintaxis de los lenguajes de programación habitualmente se describe mediante gramáticas libres de contexto (o gramáticas tipo 2).

Estas gramáticas están formadas por:

- Un conjunto finito y no vacío de símbolos terminales(o tokens): Σ.
- Un conjunto de símbolos no terminales: N.
- Un símbolo especial llamado axioma o símbolo inicial $S \in N$.
- Un conjunto de producciones que relacionan los símbolos terminales con los no terminales, de la forma: $X \to w$ con $X \in N$ y $w \in (\Sigma \cup N)^*$.

Para verificar si una instrucción pertenece al lenguaje, se construye el árbol sintáctico de la instrucción a partir de los *tokens* recibidos y que constituirán las hojas del árbol sintáctico.

Para realizar el análisis es necesario calcular previamente tres tipos de conjuntos: anulables, iniciales ($FIRST$) y seguidores ($FOLLOW$). Los anulables son los no terminales que pueden generar la palabra vacía (ϵ). El conjunto $FIRST(\alpha)$ de una cadena es el conjunto de terminales que pueden comenzar cualquier cadena obtenida a partir de α y el conjunto $FOLLOW(A)$ de un no terminal es el conjunto de terminales que pueden aparecer inmediatamente después del A. Si A es el último símbolo entonces se incluye el separador \$.

Para las gramáticas libres de contexto existen dos tipos básicos de analizadores sintácticos: los descendentes (*top-down*) y los ascendentes (*bottom-up*). Los descendentes construyen el árbol comenzando por la raíz y bajan hasta las hojas y los ascendentes construyen el árbol desde las hojas hasta la raíz.

3.3. Análisis sintáctico descendente

Los métodos descendentes tratan de generar una forma sentencial a partir del axioma reconstruyendo una derivación más a la izquierda en orden directo: $S \Rightarrow_l^* w$. El árbol de derivación se construye desde la raíz hasta las hojas (*análisis LL(k)* o *top-douw*)

Un analizador de este tipo está formado por el buffer de entrada, una pila, una tabla de análisis y el programa de control del analizador *(PDA)*.

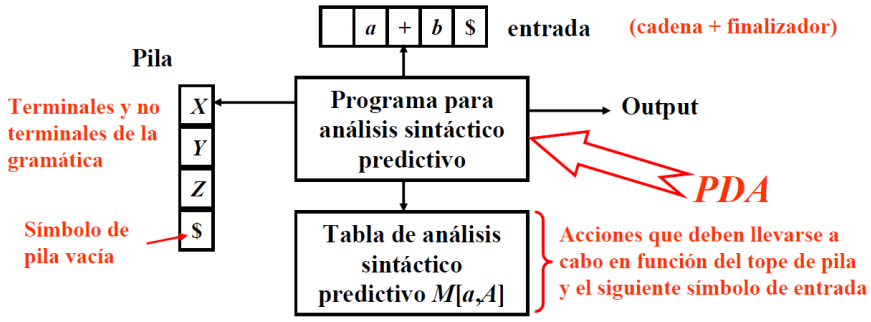

Figura 3.2: Estructura del analizador descendente [8].

El problema clave durante el análisis sintáctico es determinar la producción que debe aplicarse a un no terminal.

El analizador sintáctico no recursivo busca la producción que debe aplicarse en una tabla de análisis sintáctico predictivo $TASP$. A la vista del tope de pila y del símbolo nos dice que acción llevar a cabo.

Construcción de la *TASP*

Está compuesta por entradas de la forma $M[a, X] = (X \rightarrow w)$ cuando $a \in FIRST(w.FOLLOW(X))$.

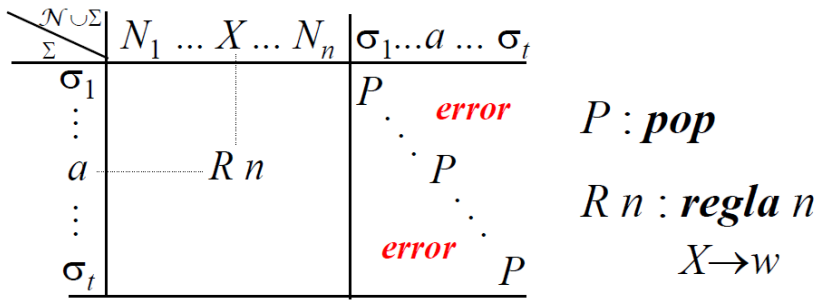

Figura 3.3: Tabla de análisis sintáctico predictivo [8].

Uso de la *TASP*

Las acciones del PDA en función de la entrada y de la pila son:

Si $X = a = \$$, cadena reconocida y se finaliza el análisis.

Si $X = a \neq \$$, hacemos un pop en la pila y llamamos al analizador léxico para obtener un nuevo token.

Si $X \neq a$, se ha producido un error, llamar a la rutina de recuperación de errores y emitir un mensaje del tipo «se esperaba un X pero se encontró un a».

En cualquier otro caso se consulta la entrada de la tabla $M[a, X]$. Debe aparecer una producción de la forma $X \rightarrow Y_1, Y_2 \ldots Y_n$. Se hace un *pop* de X y se reemplaza por $Y_1, Y_2 \ldots Y_n$ con Y_1 en el tope de la pila. Si la entrada de tabla está vacía se llama a la rutina de recuperación de error y se emite un mensaje como «X no puede comenzar por a».

Ejemplo

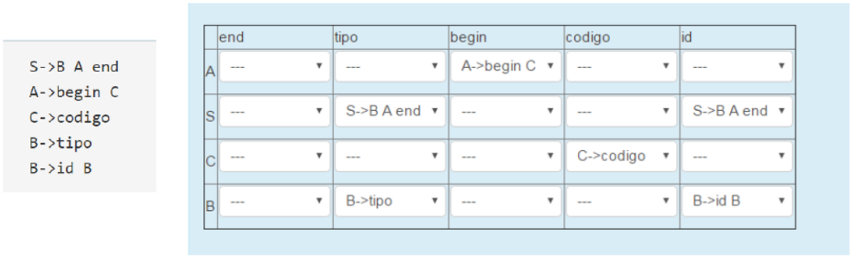

Figura 3.4: $TASP$ de la gramática.

PILA	ENTRADA	SALIDA
$, S	tipo,begin,codigo,end,$	
$, end, A, B	tipo,begin,codigo,end,$	S->B A end
$, end, A, tipo	tipo,begin,codigo,end,$	B->tipo
$, end, A	begin,codigo,end,$	B->tipo
$, end, C, begin	begin,codigo,end,$	A->begin C
$, end, C	codigo,end,$	A->begin C
$, end, codigo	codigo,end,$	C->codigo
$, end	end,$	C->codigo
$	$	

Figura 3.5: Traza de la cadena «`tipo begin codigo end $`».

Gramáticas $LL(1)$

Una gramática cuya *TASP* no tiene entradas múltiples se dice que es $LL(1)$.

Si una gramática es $LL(1)$ entonces:

- Es un gramática no ambigua (no existe ninguna cadena a la que se le pueda asociar dos árboles de análisis sintáctico diferentes).
- Es una gramática no recursiva a izquierdas (no existe ninguna derivación del tipo $A \Rightarrow^{+} A\alpha$ para un no terminal A)
- Debe estar factorizada (Para ningún terminal A hay dos producciones distintas del tipo $A \longrightarrow \alpha\beta_1$, $A \longrightarrow \alpha\beta_2$).

Se pueden modificar las gramáticas, factorizando por la izquierda o eliminando la recursividad a izquierdas sin modificar el lenguaje que se está reconociendo.

Las gramáticas $LL(1)$ permiten construir de forma automática una analizador determinista con tan solo examinar en cada momento el símbolo actual de la cadena de entrada para saber qué producción aplicar.

3.4. Análisis sintáctico ascendente

La idea es generar una forma sentencial a partir del axioma, reconstruyendo una derivación más a la derecha en orden inverso: $S \Rightarrow_r^* w$.

Los más utilizados son los analizadores sintácticos LR que utilizan la técnica de análisis por desplazamiento/reducción. Se parte de la cadena a analizar y se trata de "reducir"hasta llegar al símbolo inicial de la gramática. En cada paso de reducción se sustituye una subcadena determinada que concuerde con el lado derecho de una producción (*asidero*) por el símbolo del lado izquierdo de dicha producción.

Si en cada paso se elige correctamente la cadena, se consigue una derivación por la derecha en sentido inverso.

Las tres técnicas más comunes para construir tablas de análisis LR son:

- SLR ($LRsimple$): es la más fácil de implementar, pero la menos potente de las tres. Solo son capaces de reconocer un número reducido de gramáticas.
- LR *canónico*: es la más potente y la más costosa de las tres.
- $LALR$ ($LookaheadLR$): tiene un costo y una potencia intermedia entre los dos anteriores.

Un analizador sintáctico LR consta de un buffer de entrada, una pila, un programa conductor y una tabla de análisis sintáctico con dos partes, la de $ACCIÓN$ y la de $IR\ A$.

El programa conductor es el mismo para todos los analizadores sintácticos LR, solo cambian la forma de obtener las tablas de un tipo de análisis a otro.

El programa utiliza una pila para almacenar una cadena de la forma $s_0X_1s_1X_2s_2 \ldots X_ms_m$ donde s_m está en la cima. Cada X_i es un símbolo gramatical y cada s_i es un símbolo llamado estado.

Cada símbolo de estado resume la información contenida debajo de él en la pila, y se usan la combinación del símbolo de estado en la cima de la pila y el símbolo en curso de la entrada para indexar la tabla del análisis sintáctico y determinar la decisión de desplazamiento o reducción del analizador.

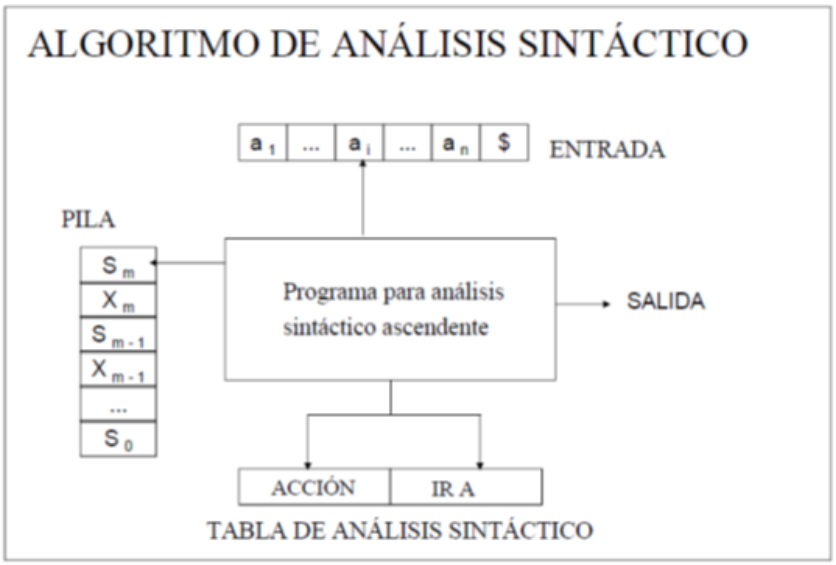

Figura 3.6: Estructura del analizador ascendente.

En cada paso hay cuatro opciones:

- Poner el token actual en la cima de la pila y llamar al analizador léxico para obtener un nuevo token ($SHIFT$).
- Decidir que los símbolos en la cima de la pila forman un asidero, y entonces desapilarlos y reemplazarlos por la parte izquierda de su producción ($REDUCE$).
- Finalizar aceptando la cadena de entrada.
- Finalizar señalando un error.

Al realizar análisis ascendente pueden surgir conflictos. Ocurre un conflicto cuando en una entrada de la tabla hay más de una acción. Hay dos tipos de conflicto $SHIFT-REDUCE$ y $REDUCE-REDUCE$ que en algunos casos se pueden resolver mirando el siguiente símbolo de la entrada.

Gramáticas LR

Una gramática para la que se puede construir una tabla de análisis LR se dice que es una gramática LR. Una gramática que puede ser analizada por un analizador LR mirando hasta k símbolos de entrada por delante (*lookaheads*) en cada movimiento, se dice que es una gramática $LR(k)$.

Análisis $SLR(1)$

Es la técnica más sencilla que permite un análisis LR con previsión de *LookAhead* de un símbolo.

Se basa en la construcción de los conjuntos de items $LR(0)$ de la gramática. Un item $LR(0)$ de una gramática es una de sus producciones, con un punto en algún lugar de su parte derecha. El punto indica hasta donde se ha visto la producción en ese momento del análisis. Los ítems se agrupan en estados de un autómata capaz de reconocer asideros en la cima de la pila.

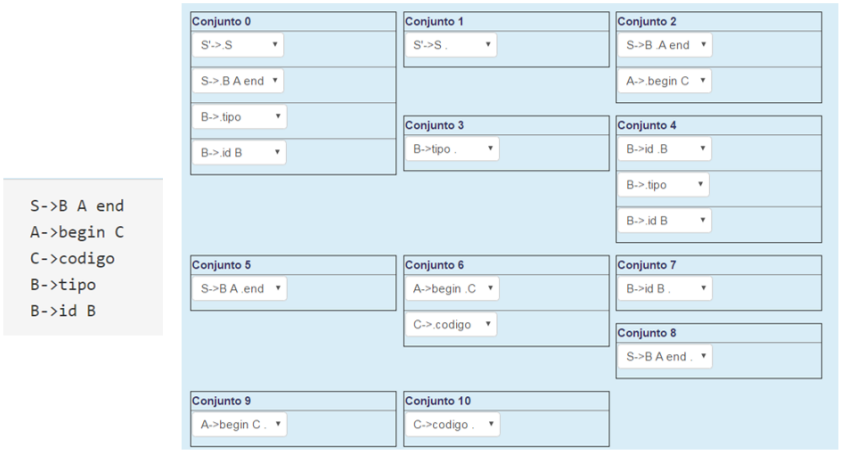

Figura 3.7: Conjuntos de ítems $LR(0)$.

Se pueden presentar conflictos *REDUCE-REDUCE* y *SHIFT-REDUCE* que en algunos casos se pueden resolver.

Análisis $LR(1)$

Se basa en la construcción de los items $LR(1)$ de la gramática. Un ítem $LR(1)$ tiene dos componentes: una producción con marca (ítem $LR(0)$) y un conjunto de símbolos de anticipación). Se aumentan los estados del autómata finito.

```
S->B A end
A->begin C
C->codigo
B->tipo
B->id B
```

Conjunto 0

Ítem marcado	Símbolos de anticipación
S'->.S	$
S->.B A end	$
B->.tipo	begin
B->.id B	begin

Conjunto 1

Ítem marcado	Símbolos de anticipación
S'->S .	$

Conjunto 2

Ítem marcado	Símbolos de anticipación
S->B .A end	$
A->.begin C	end

Conjunto 3

Ítem marcado	Símbolos de anticipación
B->tipo .	begin

Conjunto 4

Ítem marcado	Símbolos de anticipación
B->id .B	begin
B->.tipo	begin
B->.id B	begin

Figura 3.8: Primeros conjuntos de *ítems* $LR(1)$.

	Tabla de ACCIÓN						Tabla de IR A			
estado	end	tipo	begin	codigo	id	$	A	S	C	B
0	-	d3	-	-	d4	-	-	1	-	2
1	-	-	-	-	-	ACP	-	-	-	-
2	-	-	d6	-	-	-	5	-	-	-
3	-	-	r5	-	-	-	-	-	-	-
4	-	d3	-	-	d4	-	-	-	-	7
5	d8	-	-	-	-	-	-	-	-	-
6	-	-	-	d10	-	-	-	-	9	-
7	-	-	r6	-	-	-	-	-	-	-
8	-	-	-	-	-	r2	-	-	-	-
9	r3	-	-	-	-	-	-	-	-	-
10	r4	-	-	-	-	-	-	-	-	-

Figura 3.9: Tabla de ACCIÓN y de IR A en un análisis $LR(1)$.

PILA	ENTRADA	SALIDA
0	tipo,begin,codigo,end,$	
0,tipo,3	begin,codigo,end,$	
0,B,2	begin,codigo,end,$	B->tipo
0,B,2,begin,6	codigo,end,$	B->tipo
0,B,2,begin,6,codigo,10	end,$	B->tipo
0,B,2,begin,6,C,9	end,$	C->codigo
0,B,2,A,5	end,$	A->begin C
0,B,2,A,5,end,8	$	A->begin C
0,S,1	$	

Figura 3.10: Traza de análisis de la cadena «`tipo begin codigo end` $».

Análisis $LALR(1)$

El análisis $LR(1)$ *canónico* requiere muchos más estados que el análisis $SLR(1)$. Por el contrario hay algunas construcciones de los lenguajes de programación que dan problemas en el análisis $SLR(1)$.

El análisis $LALR(1)$ combina las ventajas de los dos análisis. Puede tratar con las gramáticas problemáticas para el análisis $SLR(1)$, y requiere menos estados que en el análisis $LR(1)$ *canónico*.

Una vez obtenidos los conjuntos $LR(1)$ se identifican aquellos en los que los items del núcleo sólo difieren en los lookahead y se fusionan, haciendo que su lookahead sea la unión. Sobre el nuevo conjunto de conjuntos de items marcados se aplica el mismo algoritmo de obtención de tablas *ACCIÓN* e *IR A* que para el $LR(1)$ *canónico*. Pueden surgir conflictos REDUCE/REDUCE que antes no existían.

Si surgen conflictos, la gramática no es $LALR(1)$.

Técnicas y herramientas

4.1. Técnicas

SCRUM

SCRUM[10] es una metodología ágil en la que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente y obtener el mejor resultado posible de un proyecto.

Se basa en la creación de procesos iterativos e incrementales, así se pueden realizar estimaciones de rendimiento futuro sobre las cuales se pueden tomar decisiones y controlar los riesgos.

UML[6]

Lenguaje Unificado de Modelado (Unified Modeling Language)[4] es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad en el diseño de aplicaciones Orientadas a Objetos.

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

Es importante remarcar que UML es un lenguaje de modelado para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

4.2. Herramientas

Todas las herramientas utilizadas para la elaboración del proyecto son gratuitas o se pueden obtener versiones de prueba en Internet.

[4] http://www.uml.org/

Java

Java[5] es un lenguaje de programación de propósito general, concurrente y orientado a objetos.

Se trata de un lenguaje fuerte y estáticamente tipado.

Distingue entre errores en tiempo de compilación y errores en tiempo de ejecución.

Java es un lenguaje de alto nivel, en el sentido de que los detalles de representación no son accesibles al programador. Incluye una administración automática del almacenamiento de datos en forma del recolector de basura, diseñado para evitar los problemas relacionados con la liberación manual de memoria. Es uno de los lenguajes de programación más usados en el mundo.[11]

Java SE 8 es la edición más reciente del lenguaje de programación *Java*, y representa la mayor evolución del mismo en su historia.

Incluye nuevas características, mejoras y correcciones de bugs para mejorar la eficacia en el desarrollo y la ejecución de programas *Java*.

Las librerías de la plataforma *Java* mantienen una evolución paralela a la del lenguaje.

Eclipse

Eclipse[6] es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para el desarrollo integrado (*IDE*) de aplicaciones *Java* y una base para productos basados en *Eclipse Platform*.

JUnit 4

JUnit[7] es un sistema software utilizado para realizar pruebas sobre código *Java*, formando parte de la familia de herramientas de pruebas *xUnit*. Es una de las librerías *Java* más utilizadas en proyectos de código abierto.

La versión 4 de *JUnit* extiende y simplifica la funcionalidad de anteriores versiones, haciendo un uso extensivo del sistema de anotaciones de *Java*.

Apache Maven

Maven[8] es una herramienta de administración y construcción de proyectos usada principalmente con *Java*.

[5] https://www.java.com.es
[6] https://eclipse.org/
[7] http://junit.org/
[8] http://maven.apache.org/

Su funcionamiento se basa en la existencia de un fichero de configuración XML, el *POM* (Project Object Model). Este fichero define la construcción, emisión, documentación, empaquetado, pruebas, manejo de dependencias y múltiples otras tareas de manera centralizada.

JavaCC

JavaCC[9] (*Java Compiler Compiler*) es un generador de analizadores sintácticos *Java*. Funciona mediante la especificación de una gramática en un formato propio. A partir de este fichero, la herramienta genera un programa Java capaz de procesar texto y reconocer coincidencias con la gramática.

Además del generador en sí, *JavaCC* proporciona una serie de herramientas relacionadas, como un constructor de árboles (*JJTree*) y un informe en caso de error de los mejores entre los analizadores sintácticos disponibles.

VersionOne

VersionOne[10] es un sistema de control de versiones, diseñado para trabajar con proyectos de cualquier tamaño. *VersionOne* permite mantener múltiples ramas locales independientes, cuya creación, modificación y combinado resulta poco costoso. Esto permite aislar tareas, trabajando e incorporando cada una de manera totalmente separada.

Github

Github[11] proporciona un servicio de almacenamiento de repositorios remotos y un entorno de colaboración para desarrolladores.

Github proporciona herramientas de seguimiento de proyectos, incluyendo una wiki y un sistema de seguimiento de problemas (*issue tracker*) por repositorio. También es compatible con otras aplicaciones web, como por ejemplo *Pivotal Tracker*, que facilitan su integración en el proceso de desarrollo.

Enlace al repositorio de este proyecto: `https://github.com/vrt0004/Trabajo`

Moodle

Moodle (*Modular Object-Oriented Dynamic Learning Environment*) es una plataforma de enseñanza virtual (*e-learning*) desarrollada como software libre.

Es una aplicación para crear y gestionar plataformas educativas, espacios donde un centro educativo, institución o empresa, gestiona recursos propor-

[9] `https://javacc.java.net/`
[10] `https://www.versionone.com/`
[11] `https://github.com/`

cionados por sus docentes, organiza el acceso a los mismos y permite la comunicación entre todos los implicados (alumnado y profesorado).

Una de las características de *Moodle* es su capacidad para importar cuestionarios a partir de varios formatos, incluyendo texto plano, el formato propietario *Gift* y a partir de documentos XML.

LaTeX

LaTeX[12] es un lenguaje de marcadores para la preparación de documentos comúnmente usado en publicaciones técnicas o científicas.

Es un procesador de texto «*What you see is not what you get*», lo cual significa que lo que vemos durante la edición no es el documento final, sino las instrucciones para generarlo. Esto permite separar casi completamente el contenido del documento de su formato. La salida obtenida será la misma con independencia del dispositivo o sistema operativo empleado para su visualización o impresión.

TeXMaker

TeXMaker[13] es un editor libre de LaTeX, moderno y multiplataforma para Windows, Linux y MacOSX que integra las herramientas necesarias para desarrollar documentos con LaTeX, en una sola aplicación.

TeXMaker incluye soporte *Unicode*, la corrección ortográfica, auto-completado, plegado de código y un visor de pdf con el apoyo *SyncTeX* y el modo de visualización continua. TeXMaker es fácil de usar y configurar. Se distribuye bajo la licencia *GPL*.

MiKTeX

MiKTeX es una distribución TeX/LaTeX libre de código abierto para *Windows*. Una de sus características es la capacidad que tiene para instalar paquetes automáticamente sin necesidad de intervención del usuario. Al contrario que otras distribuciones, su instalación es muy sencilla.

Mustache

Mustache[14] es un sistema de plantillas de lógica descendente para HTML, XML y otros muchos.

[12] https://latex-project.org/
[13] http://www.xm1math.net/texmaker/
[14] http://mustache.github.io/mustache.1.html

Se llama de lógica descendente porque no tiene comandos if, else, ni bucles for. En cambio, sólo se trabaja con etiquetas. Algunas etiquetas son reemplazadas por un valor, por un conjunto de valores o por nada.

XML

XML (*Extensive Markup Language*) es un lenguaje de etiquetas que se utiliza para crear documentos estructurados, compuestos de entidades que contienen en su interior datos u otras entidades. El estándar fue producido y es desarrollado por el *World Wide Web Consortium*[15].

Podemos verificar que un documento *XML* tiene el formato correcto validándolo contra un lenguaje de definición de esquemas: *DTD*[16],(*Document Type Definition*), *XML Schema*[17], *RELAX NG*[18], etc.

También podemos verificar que un documento está «bien formado», es decir, que cumple una serie de reglas gramaticales mínimas.

Commons CLI

La biblioteca *Apache Commons CLI*[19] proporciona una API para analizar las opciones de línea de comandos pasados a los programas.

También es capaz de imprimir mensajes de ayuda que detallan las opciones disponibles para una herramienta de línea de comandos.

ObjectAid UML Explorer

ObjectAid UML Explorer[20] es un complemento de visualización de código para *Eclipse*. Permite mostrar el código fuente de un proyecto Java en forma de diagramas *UML*, reflejando el estado y las relaciones en el código, y actualizándose a medida que el código cambia.

EclEmma

EclEmma[21] es una herramienta que permite examinar la cobertura de pruebas en un proyecto *Java*. Dispone un un complemento para *Eclipse* que permite realizar las comprobaciones directamente desde el *IDE*.

[15] http://www.w3.org/
[16] http://es.wikipedia.org
[17] http://www.w3.org/TR/xmlschema-0/
[18] http://www.relaxng.org/
[19] http://commons.apache.org/proper/commons-cli/
[20] http://www.objectaid.com/
[21] http://www.eclemma.org/

Jabref

Jabref[22] es un editor de referencias bibliográficas que permite introducir las referencias bibliográficas en TEXMaker de forma sencilla.

SourceMonitor

SourceMonitor[23] es una herramienta que permite obtener las métricas del codigo de la aplicación *PLGRAM*.

DIA

DIA[24] es una herramienta para dibujar diagramas de estructuras. *Dia Diagram Editor* es un software gratuito de dibujo de código abierto.

[22] http://www.jabref.org/
[23] http://www.campwoodsw.com/sourcemonitor.html
[24] https://sourceforge.net/projects/dia-installer/

Aspectos relevantes del desarrollo del proyecto

En este proyecto se presenta una aplicación de escritorio que permite generar cuestionarios sobre análisis ascendente y descendente. Además dicha aplicación permite su posterior publicación en Moodle y en formato LaTeX para su futuro uso en pruebas escritas.

5.1. Conocimientos necesarios

Para realizar esta aplicación ha sido necesario repasar y utilizar algunos de los conocimientos adquiridos durante estos últimos años en el grado.

- Estructuras de datos.

  Uno de los problemas más importantes durante el desarrollo fue la búsqueda de una estructura de datos adecuada para usar en las plantillas. Los conocimientos en estructuras de datos han servido para obtener un software más eficiente y sencillo.

- Ingeniería de software.

  Ha sido utilizada para dar un enfoque sistemático, disciplinado y cuantificable a esta aplicación.

- Gestión de proyectos.

  Se ha seguido una metodología *Scrum*. Ha sido necesario repasar los apuntes para seguirla de forma correcta.

- Procesadores de lenguajes.

  Ha sido necesario repasar toda la teoría de dicha asignatura para comprobar que los resultados obtenidos de la aplicación eran correctos.

5.2. Conocimientos adquiridos

Se han adquirido muchos conocimientos durante el desarrollo de *PLGRAM*.

- Uso de plantillas.

  Para generar los documentos se ha utilizado un motor de plantillas llamado *Mustache*.

- Archivos .XML.

  Los documentos generados con la aplicación *PLGRAM* pueden ser documentos .XML, por lo que se han adquirido conocimientos sobre su diseño y funcionamiento.

- Archivos .TEX.

  Los documentos generados con la aplicación *PLGRAM* también pueden ser documentos .TEX, por lo que se han adquirido conocimientos sobre su diseño y funcionamiento.

- Uso de TEXMAKER

  Realizar la documentación del trabajo de fin de grado con LaTeX ha sido un reto que me será útil para el futuro laboral.

- Uso de *Pluggins* en Eclipse

  La utilización de diferentes pluggins en Eclipse ha facilitado la realización de la aplicación y que el resultado sea una aplicación de calidad.

5.3. Objetivos realizados

El objetivo ha sido realizar una aplicación con una finalidad docente, que facilite la construcción de cuestionarios para poder utilizar como herramienta de aprendizaje y de evaluación en la plataforma Moodle y que también obtenera de forma sencilla documentos en formato TEX que pueden permitir, por ejemplo, realizar pruebas escritas.

Trabajos relacionados

Para comenzar a trabajar en el desarrollo de la presente aplicación se buscó información relacionada con la aplicación que se quería llevar a cabo.

Para ello, se realizó una investigación a través de la Web sobre aplicaciones que tuvieran una función similar a la desarrollada.

Aunque no existe una gran variedad de programas que realicen tareas similares a *PLGRAM*, entre los dos programas recomendados por el tutor se ha encontrado información suficiente para poder llevar a cabo la aplicación. Dichos programas son *PLQUIZ*[2] y *BURGRAM*[9].

Dado que ambos programas han sido desarrollados por alumnos de la Universidad de Burgos, como trabajos de final de grado, el tutor pudo facilitarme su estructura y toda la documentación que estimó necesaria para la correcta ejecución de *PLGRAM*.

6.1. PLQUIZ [2]

PLQUIZ[25] es una herramienta de escritorio que permite generar preguntas de test aleatorias (tipo quiz, cloze, de texto libre...) sobre problemas de algoritmos de análisis léxico. El formato utilizado para generar las preguntas es .XML para importarse a entornos virtuales de aprendizaje (Moodle), y .TEX para su impresión en papel a través de la obtención de código LaTeX.

Se han seguido sus pasos en el formato de la interfaz gráfica, pues tiene una organización muy clara para el usuario, permitiendo acceder rápidamente a las opciones disponibles.

[25] Roberto Izquierdo Amo, Julio 2014, Universidad de Burgos

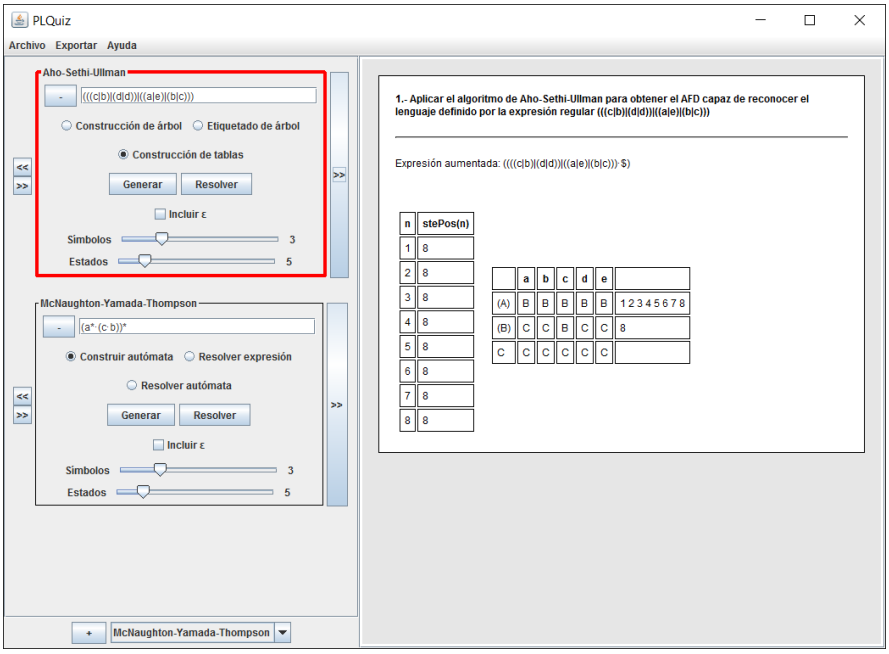

Figura 6.11: PLQUIZ

6.2. BURGRAM [9]

BURGRAM[26] es una herramienta que permite seguir paso a paso el proceso de los distintos modos de análisis sintáctico (tanto ascendente como descendente). La aplicación permite editar gramáticas, visualizar el proceso completo de operación del algoritmo sobre las mismas y la generación de informes con los resultados.

Se han reutilizado varias de las clases de este programa pues realizan el análisis que la aplicación desarrollada necesitaba para obtener los datos que posteriormente se han tratado para obtener los ficheros .XML.

[26] Carlos Gómez Palacios, Enero 2008, Universidad de Burgos

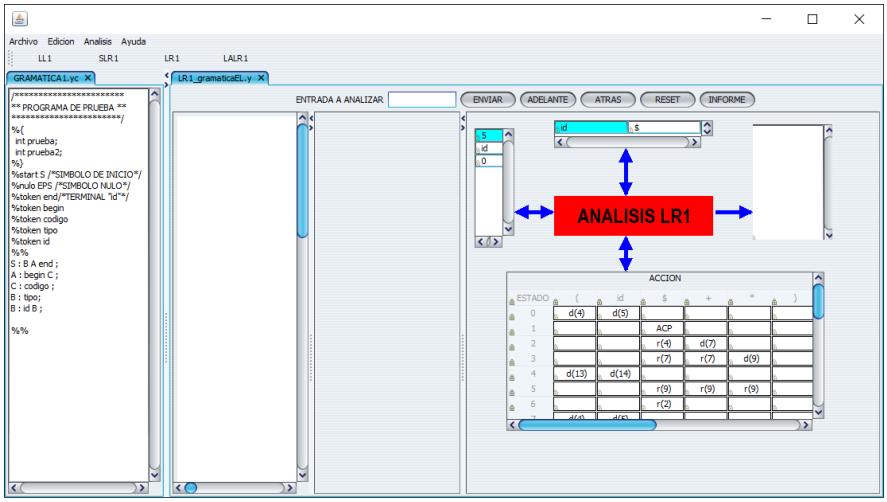

Figura 6.12: BURGRAM

6.3. SEFALAS [4]

SEFALAS[27] (**S**oftware para la **E**nseñanza de las **F**ases de **Á**nálisis **L**éxico y **A**nálisis **S**intáctico) es una aplicación desarrollada como herramienta para la enseñanza de ciertas técnicas de análisis léxico y de análisis sintáctico. En lo referente a análisis sintáctico SEFALAS recoge las dos estrategias: análisis descendente y análisis ascendente. La primera de ellas se ilustra mediante el método descendente LL(1). La segunda se ilustra con los métodos de precedencia de operador y precedencia simple y las gramáticas LR en sus diversas modalidades: SLR, LR(1) y LALR. Además, es posible escribir la gramática en formato YACC y Sefalas construye la tabla de análisis dependiendo de la estrategia seleccionada. Es posible realizar el análisis de un texto de entrada y ver la evolución del análisis de forma interactiva a través de la tabla de análisis.

[27] José Francisco Jódar Reyes, 2004, Universidad de Granada

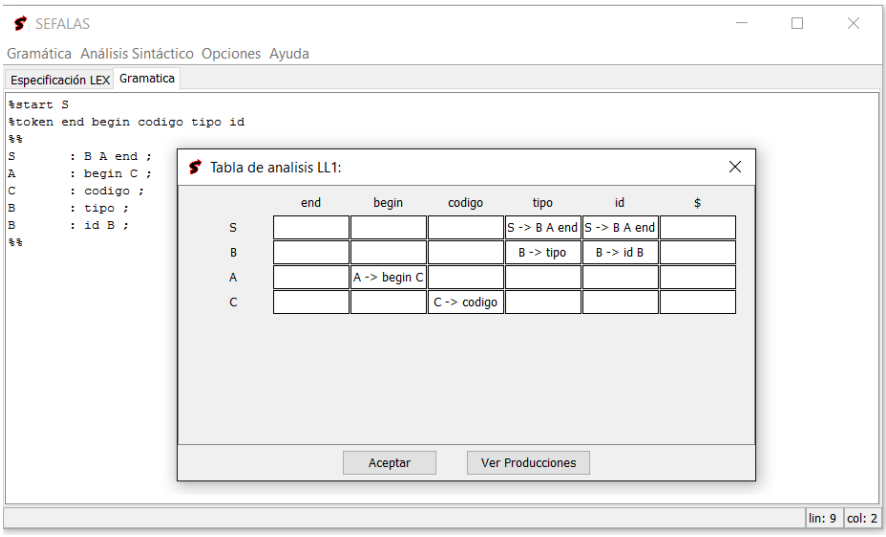

Figura 6.13: SEFALAS

6.4. ANAGRA [7]

ANAGRA[28] es una aplicación que permite editar y abrir gramáticas escritas con la sintáxis Yacc. Calcular las funciones del FIRST y FOLLOW, y otras operaciones con gramáticas.

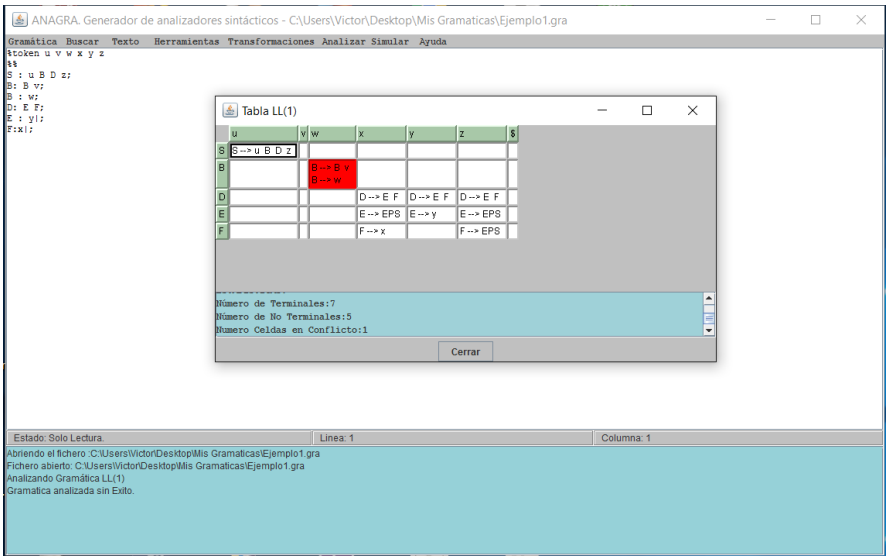

Figura 6.14: ANAGRA

[28] Raúl Novoa Mínguez,2009,Universidad de Zaragoza

6.5. Proyecto SEPa [5]

SEPa[29] es una aplicación que tiene como objetivo facilitar al estudiante el aprendizaje autónomo de conceptos y procedimientos que realizan los compiladores.

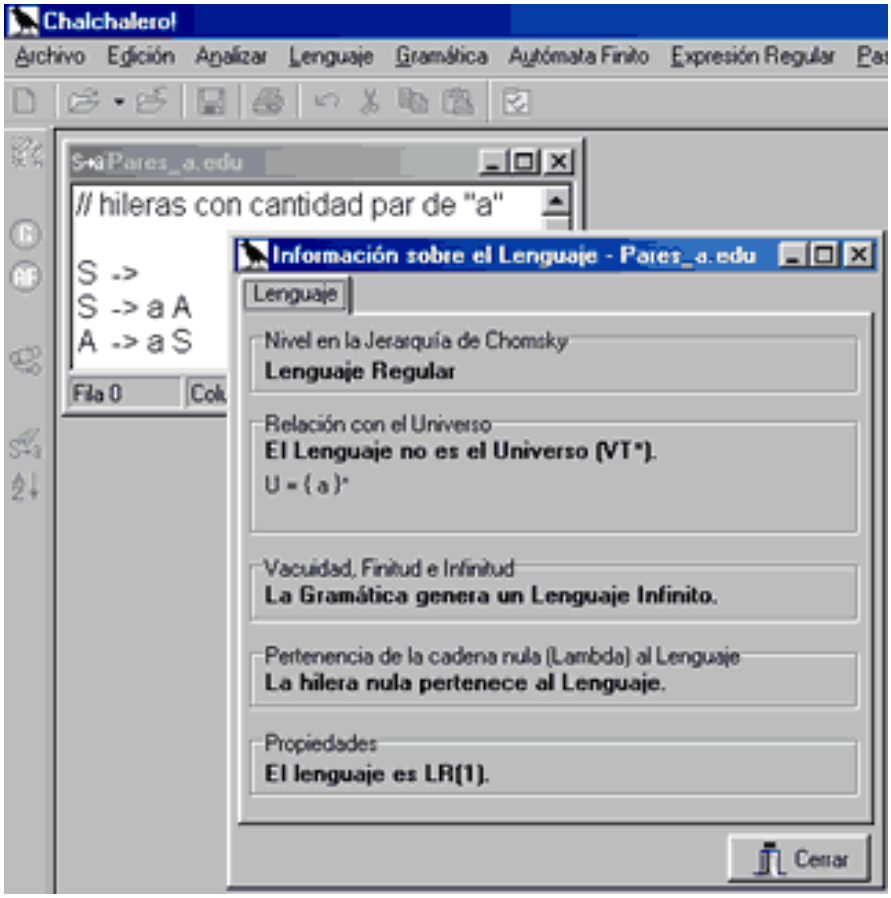

Figura 6.15: SEPa

[29] Juan José Tamagnini,Salvador Valerio Cavadini y Pablo Luis Berdaguer,2003 ,Universidad católica de Santiago del Estero

Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

Se ha perfeccionado el conocimiento sobre Java y su integración con bibliotecas de terceros como ha sido *Commons CLI*.

El empleo de una herramienta para el control de versiones, ya utilizados con anterioridad, ha servido para reforzar la idea de lo imprescindibles que son estas herramientas para el desarrollo software y las garantías de seguridad que ofrecen.

Se ha aprendido un nuevo modo de realizar documentos técnicos con el uso de LaTeX. Aunque en un principio supuso una carga a la documentación, a medida que avanzó el desarrollo, favoreció el interés por la escritura debido a los retos que se presentaron durante su ejecución.

Por último, destacar el perfeccionamiento de todas las tareas del desarrollo software aprendidas a lo largo del grado en ingeniería: planificación, análisis, diseño, implementación, pruebas y documentación.

7.2. Líneas de trabajo futuras

Deshacer cambios

La aplicación no permite *volver atrás* si se elimina parte del trabajo realizado. Por ejemplo, si se elimina una cuestión, no se puede recuperar.

Desarrollar una opción de «deshacer» evitaría problemas al usuario y aumentaría la usabilidad del programa.

Exportación de más de una pregunta en la interfaz gráfica

La aplicación exporta las preguntas en formato .XML o .TEX de una en una.

Una posible mejora de la aplicación consistiría en poder exportar todas las preguntas a un mismo fichero facilitando el trabajo y reduciendo el tiempo de uso.

Más tipos de cuestiones

La aplicación proporciona un conjunto relativamente amplio de modelos de preguntas, pero, partiendo de la funcionalidad de resolución de algoritmos ya implementada, podrían generarse nuevos tipos de cuestiones para evaluar conocimientos de manera más amplia.

Un ejemplo serían preguntas de emparejamiento, que están directamente soportadas por *Moodle* y son sencillas de crear en otros formatos

Importar cuestionarios

En la versión actual de la aplicación, los cuestionarios pueden exportarse pero no importarse. Esto quiere decir que, una vez se cierra el programa, no se puede continuar trabajando sobre las preguntas que se tenían, a menos que se cree un cuestionario nuevo y se añadan las preguntas a mano.

Se proponen dos posibles implementaciones de un sistema de almacenamiento y carga:

- El almacenamiento de los cuestionarios como ficheros separados, ya sea en texto plano, XML o cualquier otro formato.
- Consistiría en la lectura de un fichero exportado y su comparación con la plantilla de la que se generó. De esta manera se pueden localizar la gramática y el análisis y extraer la plantilla. Una vez obtenidas, la reconstrucción del cuestionario es trivial.

Bibliografía

[1] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. *Compilers, Principles, Techniques*. Addison wesley, 1986.

[2] Roberto Izquierdo Amo. *PLQuiz*. Universidad de Burgos, Julio 2014.

[3] Jose Antonio Jimenez Jimenez Millan. *Compiladores y procesadores de lenguajes*. Number 005.13. 2009.

[4] JF Jodar-Reyes and J Revelles-Moreno. Sefalas: Software para la enseñanza de las fases de análisis léxico y análisis sintáctico, 2008.

[5] Salvador Valerio Cavadini, Pablo Luis Berdaguer y Juan José Tamagnini. Proyecto sepa. SEPa, 2003. Universidad católica de Santiago del Estero.

[6] Craig Larman. *UML y Patrones*. Pearson, 1999.

[7] Raúl Novoa Mínguez. Anagra, jan 2009. Universidad de Zaragoza.

[8] César Garcia Osorio. Apuntes procesadores de lenguajes.

[9] Jesús Maudes Raedo and Juan José Rodríguez Diez. Burgram: Una herramienta interactiva para el estudio de los algoritmos de análisis sintáctico ascendente y descendente.

[10] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum Alliance*, 2011.

[11] Universia. Los 10 lenguajes de programación más populares en la actualidad, feb 2016.

[12] Wikipedia. Historia de la construcción de los compiladores, nov 2015.